AMENDMENTS TO THE DRAWINGS

Enclosed are new Formal Drawings that now identify Figures 1 and 2 as Prior Art, as requested by the Examiner. Each drawing is properly identified in the top margin as "Replacement Sheet," as required by 37 CFR 1.121(d).

REMARKS

Claims 1-31 are pending in the current application. In an Office Action dated February 28, 2006 ("Office Action"), the Examiner required designation of Figures 1 and 2 as --Prior Art--, objected to Figure 3, required correction of the figure captions with respect to Figure 2 and Figure 5, objected to the specification under 37 C.F.R. § 1.75(d)(1) and M.P.E.P. § 608.01(o), rejected claims 21-31 under 35 U.S.C. § 101, rejected claims 1-2, 4, 13, 21-22, and 24 under 35 U.S.C. § 102(b) as being anticipated by "Inside Windows NT 2d. Edition," David A. Solomon ("Solomon"), rejected claims 5, 8-9, 12, 14, 17, 20, 25, 28-29, and 31 under 35 U.S.C. § 102(b), as being anticipated by Solomon in view of an email exchange obtained at the Internet address <http://sqid-cache.org/mailarchive/sqi-users/199708/0124.html> ("Wemm") provided in accordance with M.P.E.P. § 2131.01(II), rejected claims 2 and 22 under 35 U.S.C. § 102(b) as anticipated by, or, in the alternative, as obvious under 35 U.S.C. § 103(a) over, Solomon, rejected claims 6-7, 15-16, and 26-27 under 35 U.S.C. § 103(a) as being obvious over Solomon in view of Wemm, the latter being provided in accordance with M.P.E.P. § 2131.01(II) and further in view of "Structured Computer Organization 2d. Edition" by Andrew S. Tanenbaum ("Tanenbaum"), rejected claims 10, 18, and 30 under 35 U.S.C. § 103(a) as being obvious over Solomon in view of Wemm and further in view of LeClerg, U.S. Patent No. 6,857,041 B2 ("LeClerg"), and rejected claims 11 and 19 under 35 U.S.C. § 103(a) as being obvious over Solomon in view of Wemm and further in view of Liew, U.S. Patent No. 6,665,249 ("Liew").

Applicant's representative has provided formal drawings in which the requested prior art labels have been added to Figures 1 and 2. Applicant's representative respectfully traverses the objection to Figure 3. Applicant's representative has amended the specification, as required by the Examiner, to address the problem with figure captions related to Figures 2 and 5. Applicant's representative respectfully disagrees with the Examiner's objection to the specification with respect to claim 29. Applicant's representative respectfully traverses the 35 U.S.C. § 101, 35 U.S.C. § 102, and 35 U.S.C. § 103 rejections of claims 1-31.

Applicant's representative observes that the Office Action is more than twice as long, in text pages, than the current application. While Applicant's representative appreciates the Examiner's effort and time spent on the Office Action, Applicant's representative feels that the Examiner has failed to fully appreciate and understand the currently claimed invention. While the cited references are certainly relevant to the general topic of computer architecture and computer operating systems, they do not disclose, teach, mention, or even suggest the currently claimed invention. Rather than respond point-by-point to the rather lengthy claim rejections, Applicant's representative, in the interest of brevity, instead summarizes, in following subsections, the current application and claims and the cited references. Applicant's representative then contrasts the currently claimed invention to what is disclosed in the cited references, in a final subsection.

The Currently Claimed Invention

As set forth in the Technical Field section of the current application, the currently claimed invention is "a method and system for efficiently providing default values to software without allocating or initializing the memory pages and without occupying space in the memory hierarchy." The method and system of the current claims are referred to as providing "immediate virtual memory" by Applicant. Immediate virtual memory is a new type of virtual memory made possible by features of the Intel Itanium® architecture and other modern processor architectures and by the currently claimed invention.

Currently available virtual memory systems provided and supported by currently available operating systems are described in the Background of the Invention section of the current application, beginning on line 16 of page 1, with reference to Figures 1 and 2. In this background section, the virtual-to-physical-address translation mechanism employed in a wide variety of modern computer systems, and implemented in a wide variety of currently available computer operating systems, is described in detail. In general, virtual memory systems provide much larger memory address spaces to the processes executing on a computer system than the physical memory resource available

within the computer system. This is accomplished by translating virtual memory addresses to physical memory addresses and paging data into physical memory from much higher capacity mass-storage devices and out of physical memory to the much higher capacity mass-storage devices as needed to give the illusion of the larger, virtual-memory address space to processes executing within the computer system. The background section discusses translation look-aside buffers ("TLB"), a set of specialized, high-speed registers that store most recently accessed virtual-page translations, in the paragraph beginning on line 14 of page 2.

As discussed beginning on line 25 of page 3 of the current application:

When an operating system or application program begins execution, or when additional memory is allocated for operating system or program use, a computer system commonly allocates a large number of virtual pages on behalf of the operating system or application program. In many cases, the operating system or application program expects that newly allocated virtual pages are initialized to a default value. . . . In many currently available computer systems, the newly allocated virtual memory pages are fully instantiated, meaning that, when sufficient physical memory is available, a physical memory page corresponding to the virtual memory page is assigned for the virtual memory page and that the physical memory page is initialized to the default value.

Figure 2 illustrates the above-mentioned allocation of virtual memory pages on behalf of a process. As shown in Figure 2, the virtual memory pages are allocated in memory, TLB entries corresponding to the virtual memory pages are placed into the TLB, and the physical pages corresponding to the virtual memory pages are initialized to have a default value. In the case shown in Figure 2, the default value is "0." As pointed out in the paragraph beginning on line 4 of page 4, "[i]n general, virtual memory pages are either immediately initialized, under program control, as part of the virtual-page allocation process, or selected from a pool of pre-initialized pages that are zeroed or otherwise initialized in a background, operating-system process." As further pointed out in that paragraph, the initialization of virtual-memory pages is computationally expensive and may involve significant time delays. In the paragraph beginning on line 17 of page 4, Applicant points out that because of the computationally expensive and potentially slow initialization of virtual-memory pages in currently available operating systems,

manufacturers, designers, and users of computer systems have all recognized the need for systems and methods that efficiently initialize newly allocated virtual-memory pages.

As pointed out in the current application beginning on line 16 of page 7, while the value "0" is most commonly considered the default value for newly initialized virtual-memory pages, other initializations may be desirable. Two examples are then provided, the value "1" and initialization to randomly generated numbers. A third example is provided beginning on line 2 of page 8, in which all bytes within a virtual memory page are initialized to the binary value "0010." On line 17 of page 8, the application makes clear that additional default initializations are possible. Additional examples mentioned beginning on line 4 of page 8 include algorithmically generated numeric patterns, environmental variables, and other such values. Thus, it is abundantly clear in the current application that a large variety of different repeated-single-value and more complex, computed-value initializations are contemplated as default values provided by the currently claimed immediate virtual memory.

Immediate virtual memory is well summarized in the summary of the invention section of the current application:

> One or more bit flags within each translation indicate whether or not a corresponding virtual memory page is immediate. READ access to immediate virtual memory is satisfied by hardware-supplied or software-supplied values. WRITE access to immediate virtual memory raises an exception to allow an operating system to allocate physical memory for storing values written to the immediate virtual memory by the WRITE access.

To further summarize, immediate virtual memory only allocates physical memory in the case of a WRITE access. If the virtual memory is only accessed for READ operations, then no physical memory need be allocated, since the values returned to the accessing entity as a result of a READ operation are generated either in hardware or by executable software routines. In other words, the default value or values to which an immediate virtual memory page is initialized are not stored in physical memory, but are instead generated, on each access, by hardware or by executable software routines unless a WRITE access is directed to the virtual memory page.

Figure 3 of the current application illustrates a logical mechanism

embodied in processor logic that implements immediate virtual memory. In Figure 3, the virtual page number 310 within a virtual address 308 is used to access the TLB entry 302 corresponding to the virtual address. The TLB entry includes an immediate bit flag 304. When the virtual address is accessed by a process, the processor checks, in step 312 in the control-flow diagram included in Figure 3, whether or not the immediate bit 304 is set. If the immediate bit is not set, then the processor accesses the page through the standard virtual-memory hierarchy, in step 314. However, if the immediate bit is set, as determined in step 312, then, in step 316, the processor determines whether or not the access to the virtual address is a WRITE or READ access. In the case of a READ access, default data is returned, in step 320. As discussed above, the default data is generated either by hardware or by an executable software routine; it is not actually stored within the memory system of the computer. No physical memory needs to have been allocated for the page in order to fully execute READ access to an immediate virtual memory address. By contrast, if the access is a WRITE access, as determined in step 316, then the processor generates an uninstantiated page exception, in step 318, which invokes exception handling by the operating system to allocate and access the page, as discussed in the current application beginning on line 17 of page 6, which eventually leads to the processor accessing the page through the standard virtual-memory hierarchy, in step 314..

Again, to summarize, immediate virtual memory is virtual memory that is physically allocated by a memory system only in the case that the memory is accessed for a WRITE operation. Immediate virtual memory can be successfully and repeatedly accessed for READ operations without physical allocation and initialization of pages within memory, because immediate virtual memory is considered to have been initialized to a default value, and that default value can be generated in hardware or programmatically by an executable software routine, rather than being fetched from initialized pages within the memory system.

<u>Solomon</u>

Solomon, by contrast to the virtual-memory system disclosed in the current application, does not provide immediate virtual memory. Solomon discusses a

very standard, conventional operating system with conventional virtual-memory mechanisms and page-fault handling. For example, in Table 513 on pages 265-266 of Solomon, Solomon lists the various types of page faults handled by the Windows NT operating system:

Table 5-13 Reasons for Access Faults

Reason for Fault	Result
Accessing a page that is not resident in memory but is on disk in a page file or mapped file	Allocate a physical page and read the desired page from disk and into the working set
Accessing a page that is on the standby or modified list	Transition the page to the process or system working set
Accessing a page that has no committed storage (for example, reserved address space or address space that is not allocated)	Access violation
Accessing a page from user mode that can be accessed only in kernel mode	Access violation
Writing to a page that is read-only	Access violation
Accessing a demand-zero page	Add a zero-filled page to the process working set
Writing to a guard page	Guard-page violation (if a reference to a user-mode stack, perform automatic stack expansion)
Writing to a copy-on-write page	Make process-private copy of page and replace original in process or system working set
Referencing a page in system space that is valid but not in the process page directory (for example, if paged pool expanded after the process page directory was created)	Copy page directory entry from master system page directory structure and dismiss exception. (This fault is never pointed to by hardware.)
On a multiprocessor system, writing to a page that is valid but hasn't yet been written to	Set dirty bit in PTE

Inspection of this table reveals that there is no page fault in Windows NT for implementing immediate virtual memory. Windows NT does have a page fault corresponding to accessing a "demand-zero page." In response to that page fault, the operating system adds a zero-filled page to the process working set. This is described, in more detail, on page 266 of Solomon under the bulleted heading "Demand Zero:"

> The desired page must be satisfied with a page of zeroes. The pager looks at the zero page list. If the list is empty, the pager takes a page from the standby list and zeros it. The PTE format is the same as the page file PTE shown above, but the page file number and offset are zeros.

As clearly stated by Solomon, the operating system actually zeros a physical page in handling a "demand-zero" page fault. There is no mention in Solomon of generating default values by hardware or by software-executable routine for return to an accessing process, rather than allocating and initializing an actual physical memory page. Solomon does not in any way distinguish between READ and WRITE access to demand-zero pages.

On pages 261-262 of Solomon, Solomon describes a standard, conventional translation look-aside buffer. The TLB entries described in Solomon include data portions that contain a physical page number, protection field, valid bit, and usually a dirty bit, indicating the condition of the page to which the cache PTE corresponds. Solomon does not teach, mention, or even suggest an immediate-virtual-memory bit, which is not surprising, since Solomon does not teach, mention, or suggest immediate virtual memory.

Finally, Solomon describes virtual address descriptors on pages 273-275, as follows:

The memory manager uses a demand-paging algorithm to know when to load pages into memory, waiting until some thread references an address and incurs a page fault before retrieving the page from disk. Like copy-on-write, demand paging is a form of lazy evaluation--waiting to perform a task until it is required.

The memory manager uses lazy evaluation not only to bring pages into memory but also to construct the page tables required to describe new pages. For example, when a thread commits a large region of virtual memory with *VirtualAlloc*, the memory manager could immediately construct the page tables required to access the entire range of allocated memory. But what if some of that range is never accessed? Creating page tables for the entire range would be a wasted effort. *Instead, the memory manager waits to create a page table until a thread incurs a page fault, and then it creates a page table for that page.* This method significantly improves performance for processes that reserve and/or commit a lot of memory but access it sparsely.

With the lazy-evaluation algorithm, allocating even large blocks of memory is a fast operation. This performance gain is not without its trade-offs, however: when a thread allocates memory, the memory manager must respond with a range of addresses for the

> thread to use. Because the memory manager doesn't build page tables until the thread actually accesses the memory, it can't look there to determine which virtual addresses are free. To solve this problem, the memory manager maintains another set of data structures to keep track of which virtual addresses have been reserved in the process's address space and which have not. These data structures are known as virtual address descriptors (VADs). For each process, the memory manager maintains a set of VADs that describes the status of the process's address space. VADs are structured as a self-balancing binary tree to make lookups efficient. A diagram of a VAD tree is shown in Figure 5-15 on the following page.
>
> When a process reserves address space or maps a view of a section, the memory manager creates a VAD to store any information supplied by the allocation request, such as the range of addresses being reserved, whether the range will be shared or private, whether a child process can inherit the contents of the range, and the page protection applied to pages in the range.
>
> When a thread first accesses an address, the memory manager must create a PTE for the page containing the address. To do so, it finds the VAD whose address range contains the access address and uses the information it finds to fill in the PTE. If the address falls outside the range covered by the VAD, the memory manager knows that the thread did not allocate the memory before attempting to use it and therefore generates an access violation. (emphasis added)

These virtual-address descriptors are used to implement lazy evaluation in bringing pages into memory and constructing page tables required to describe the pages. As is commonly done in most currently available operating systems, the memory manager waits to create a page table and allocate a page until the page is actually accessed by a process. As explicitly stated by Solomon, when a thread first accesses an address, the memory manager creates a page-table entry for the page and allocates the page. Solomon does not in this passage, or in any other passage, distinguish between READ and WRITE access. Thus, Solomon explicitly states that, upon any first access, including a first READ access, a page table entry is created, and a physical memory page allocated, for the accessed virtual-memory address. There is nothing in Solomon to suggest physical allocation only for WRITE accesses, and generating default values in hardware or

programmatically for READ access. In other words, Solomon does not teach, mention, or suggest immediate virtual memory, and is therefore unrelated to the currently claimed invention.

Wemm

Apparently Wemm was cited for the mention of demand zeroing of pages. Wemm explains demand zeroing as follows:

> The pages are not "really" attached to your process yet, but when you access them for the first time, the page fault causes the page to be connected to the process address base and zeroed - this saves a necessary zeroing of pages that are allocated but never used.

This is entirely and exactly the demand-zeroing of pages described in Solomon, discussed in the previous subsection. Note that, whenever a page is accessed, regardless of whether the access is a READ access or a WRITE access, a page fault is generated causing the page to be allocated and zeroed. This is not the same as immediate virtual memory, in which READ accesses do not cause page faults in page allocation, but instead elicit the return of hardware-generated or executable-software-routine-generated default values that are returned to the accessing entity. Like Solomon, Wemm is unrelated to the currently claimed invention.

LeClerg

LeClerg is apparently cited by the Examiner for the proposition that physical memory can be initialized to any suitable value. Applicant's representative agrees that physical memory can be initialized to any suitable value. Initialization of physical memory is well known, and has been for at least 50 years.

Tanenbaum

Tanenbaum appears to be cited by the Examiner for the proposition that hardware and software are logically equivalent and therefore any operation performed by software can also be built directly into hardware. While theoretically true, Tanenbaum also qualifies his statement as follows:

> The decision to put certain functions in hardware and others in
> software is based on such factors as cost, speed, reliability, and
> frequency of expected changes.

In other words, while it is theoretically true that an arbitrarily complex hardware design can provide the same functionality as an arbitrary software program, it would be, for example, commercially and practically impossible to implement a modern, complex, high-level-language compiler, such as a C++ compiler, in hardware. First, the required logic circuitry would be incredibly complex and expensive to design and fabricate. More importantly, compilers are frequently modified and extended, and often contain myriad bugs and logic flaws that are not revealed until the compiler is actually used. For rapidly changing logical entities, software is the only practical implementation medium, because it can be easily modified and enhanced. For these reasons, it is impractical to implement a modern C++ compiler in hardware. However, Applicant's representative does agree, in principle, that software and hardware are interchangeable. This fact has been known for at least 50 years.

Liew and Sakakura

Liew and Sakakura are cited by the Examiner, as LeClerg, for the proposition that physical memory may be initialized to have a randomly generated value or value "-1." As with LeClerg, Applicant's representative appreciates that physical memory can be initialized to any value. As with LeClerg, this has been well known for at least 50 years. However, Liew explicitly states that a pseudo-random number generator is used to generate the random data, rather than a true random-number-generating process, such as using signal noise. Thus, Liew does not quite teach that for which it is cited.

Sloane

Sloane is cited by the Examiner for the proposition that it is obvious to represent the value "-1" as a two's complement number. Since two's complement representation of digital values has been used for at least 50 years in computer hardware,

Applicant's representative happily acknowledges that it is quite obvious to represent the value "-1" in two's complement arithmetic.

Popov

In the Notice of References Cited, the reference "Inflatable Arrays in Win32" by Petko Popov is cited. This reference, like Solomon and Wemm, is unrelated to the currently claimed invention. Popov introduces the idea of uncommitted virtual memory guarded by an exception handler. When a location within the memory is accessed, the fault handler then commits the memory. There is no discussion in Popov of committing memory only for WRITE accesses, and returning hardware-generated or software-generated values in the case of READ access, without committing memory. Popov, like Solomon and Wemm, does not teach, mention, or suggest immediate virtual memory.

Response to the Examiner's Rejections

As discussed above, the current application and current claims are directed to a new type of virtual memory referred to as "immediate virtual memory." Immediate virtual memory is fully and completely summarized in the summary of the invention section of the current application, as follows:

> Various embodiments of the present invention provide for immediate allocation of virtual memory on behalf of processes running within a computer system. One or more bit flags within each translation indicate whether or not a corresponding virtual memory page is immediate. READ access to immediate virtual memory is satisfied by hardware-supplied or software-supplied values. WRITE access to immediate virtual memory raises an exception to allow an operating system to allocate physical memory for storing values written to the immediate virtual memory by the WRITE access.

As also discussed above, none of the cited references teach, mention, or suggest immediate virtual memory. The cited references refer to common, conventional operating system techniques, including techniques for providing on-demand zeroed pages. However, in all of these techniques and discussions, no distinction is made

between READ access and WRITE access. In all of these systems, whenever a process accesses uninstantiated virtual-memory page, whether by READ access or WRITE access, a page fault is generated, and the page is instantiated in memory. Independent claims 1, 13, and 21 specifically recite "immediate virtual memory." Immediate virtual memory is clearly and thoroughly defined in the current application. An implementation of immediate virtual memory is provided, beginning on line 24 with page 5, with reference to Figure 3. Because all of the independent claims of the current application specifically recite "immediate virtual memory," and because none of the cited reference teaches, mentions, or suggests, alone or in combination, immediate virtual memory, none of the cited references individually, or in combination, anticipate or make obvious any of the claims of the current application. Moreover, the term "immediate virtual memory" was specifically selected by the Applicant to describe embodiments of the currently claimed invention because the term has not previously been used to describe a type of virtual memory.

Applicant's representative disagrees with the Examiner that Figure 3 omits a step in processing. Exception handling by operating systems is well known. An uninstantiated page exception is raised in step 318. Page exceptions are handled by operating systems which, during handling of page exceptions, invoke processor support, in step 314, for translating virtual-memory addresses. The flow diagram shown in Figure 3 is not intended to diagram the entire workings of a computer system, including the operating system, but instead is intended to describe processor support for immediate virtual memory, as explicitly stated in the caption for Figure 3: " Figure 3 shows a logical mechanism embodied in *processor logic* for implementing uninstantiated virtual memory pages according to one embodiment" (emphasis added). Processors do not handle uninstantiated page exceptions in the described implementation. There is no omitted step.

Applicant's representative disagrees with the Examiner that claim 29 lacks proper antecedent basis. As discussed in detail, above, the current specification makes it abundantly clear that any repeated single value, or computable value, can be used for default initialization of a page, depending on the desire of the computer-system architect.

In fact, the Examiner argues that it is obvious to represent the value "-1" in two's complement arithmetic. In fact, it is. The value "-1" is an often-used, special value, like the value "0" and "1." As one example, the value "-1" is used as a special return value in many C, C++, and Unix library routines. Moreover, claim 29 is part of the specification, since it was submitted as an original claim with a specification.

With regard to the Examiner's 35 U.S.C. § 101 rejection of claims 21-31, the current application makes it clear that initialization of virtual memory pages in currently available, conventional operating systems is both time and computationally expensive. The current application clearly indicates that the current invention is motivated by desire to more efficiently initialize virtual memory pages. The current application clearly and unambiguously states that any of a large variety of different default values may be desired as the default values used to initialize virtual memory pages. Thus, the current application clearly and precisely defines the reasonable motivation for immediate virtual memory - a type of virtual memory that is initialized to an arbitrary, desired default value by a process that is more efficient in both time and computational overhead than conventional virtual-memory-page initialization methods embodied in currently available operating systems.

In Applicant's representative's opinion, all of the claims remaining in the current application are clearly allowable. Favorable consideration and a Notice of Allowance are earnestly solicited.

Respectfully submitted,
John S. Worley
Olympic Patent Works PLLC

Robert W. Bergstrom
Registration No. 39,906

Enclosures:
    Postcards (2)
    Transmittal in duplicate

Olympic Patent Works PLLC
P.O. Box 4277
Seattle, WA 98194-0277
206.621.1933 telephone
206.621.5302 fax